appears in Proceedings of the Eurographics Workshop on Animation and Simulation Vienna, Austria, September 1991 pages 223-243

# Programming Mechanical Simulations

Joseph K. Kearney, Stuart Hansen, and James F. Cremer



## Abstract

This paper examines the control of complex physical objects in simulation. We introduce a programming paradigm that allows a simulation to be treated as a multi-level constraint solver. The control programmer is given the ability to specify constraints on the controlled response of mechanisms and to conditionally change these constraints dependent on the state of system. The approach facilitates the development of model-based, event-driven control programs. The usefulness of the paradigm is demonstrated through the simulation of a hopping robot.

**Keywords:** animation, constraints, dynamic simulation, motion control, robot locomotion.

## 1. Introduction

Computer simulation promises to have a dramatic influence on the study of mechanical systems and the interaction between humans and machines. The increasingly widespread use of dynamic simulation in computer-aided design, mechanical analysis, and computer animation has created new demands for control capabilities [2-6,18,19,26,28,29,31]. Most simulation systems provide only rudimentary means to program controlled motions. Many of the interesting problems in design and animation involve the simulation of complex, adaptive mechanisms such as walking robots or dexterous robot hands. Flexible, open systems are needed to allow experimentation with control methodologies for these problems.

This paper examines mechanical simulation as a programming problem. We will focus on tools that allow the user to define the controlled response of devices and the state dependent interactions between objects. A conceptual framework for programming mechanical simulations is presented. The programming paradigm is based on a view of the simulator as a multi-level constraint solver. Simulation is cast as a problem of solving a sequence of dynamic constraints on the attributes of models of physical objects. Constraints due to mechanical considerations, such as joints between bodies, and constraints due to control considerations, such as guided movements, are treated in a uniform fashion.

The set of motion constraint equations is derived from three sources. First, a base set of equations modeling the fundamental laws of motion and the mechanical connections between objects is automatically formulated from a model definition. Secondly, the base set of equations may be reformulated during the course of a simulation to accommodate changing relationships among objects as new contacts are made and old contacts are broken. Lastly, control routines may influence the composition of the set of motion equations. The user conducting the simulation models control constraints by programmatically adding and deleting named equations to the set of motion equations.

In many simulation systems including ADAMS, DADS, and SD/FAST [7,9,13], motion constraints are predetermined by a model of the mechanism and remain unchanged during simulation. By allowing conditional reformulation of motion constraints and by incorporating control constraints into the set of motion equations, the power and expressiveness of

simulation programs is significantly extended. However, the constraint restructuring must be accomplished in a carefully disciplined manner to avoid compromising the integrity of the simulation. We present a regimen under which the flexibility of constraint editing is achieved without disrupting the basic integration process.

In section 2, mechanical simulation is presented in the constraint programming framework. Higher order constraints are introduced as a means to accommodate changing relationships between objects in section 3. Constraint-based control programming is introduced in section 4. In section 5, event-driven control programming is illustrated with repetitive hopping.

## 2. Mechanical Simulation

Mechanical simulation can be viewed as a process of iterative constraint satisfaction. To a first approximation, most mechanical simulators proceed as follows:

(1) Given positions, velocities, and forces acting on all objects, a set of constraint equations governing the motions of objects are solved to determine accelerations.

(2) New positions and velocities are determined by integrating the accelerations over a small time interval.

(3) The simulation clock is incremented and the process is repeated.

The nature of the motion constraints that can be expressed, the equation solving algorithm, and integration method vary considerably across simulation systems. In our presentation, we will try to focus on characteristics shared by a large number of systems. We will illustrate points with the Newton simulator [8,12]. In this section, we consider simulation of uncontrolled devices with a constraint set that remains unchanged during the simulation.

To simulate the motion of a mechanical device, we first need a model describing its component bodies and the constraints on the motions of bodies. In Newton, the primitive objects are rigid solids. Primitive objects are joined by hinges to form composite objects. A hinge constrains the relative motion of two objects. As an example of a simple, composite object we consider a pendulum consisting of a cuboid attached to a fixed base by a spherical hinge. The definition of the pendulum model in Newton's model description language is shown in Fig. 1.

The model must contain sufficient information to determine the mass and inertial properties of the bodies and the constraints on body motions. A simulation system should provide a library of hinges that impose a variety of motion constraints. Common hinges include bracket, translational, revolute, universal, spherical, and planar. Each hinge constrains some combination of relative motions between two objects or between an object and an inertial reference frame. The spherical hinge in the pendulum example constrains a point on the pendulum to be coincident with a point on the base.

From the model definition, a system of time-varying linear equations constraining the instantaneous motion of every body can be derived. These motion equations relate accelerations of bodies to the forces and torques on the bodies dependent on object positions and velocities. Given the positions and velocities of all bodies at time $t$, positions and velocities at time $t + \Delta t$ are found by solving the motion equations for accelerations and integrating from previous states. Given initial positions and velocities, this process is iterated to simulate motion over an interval of time.

Even very simple models can lead to large, complex sets of motion equations. All of the popular mechanical simulators relieve the user from the responsibility of deriving these equations [7-9,12,13]. These systems provide a modeling interface that allows a user to

```
hinge ball.and.socket;

primitive rod (x,y,z)
 begin
  properties: (density: 1.0, color: 'red');
  geometry: cuboid(x,y,z)
 where begin
                top_hingepoint: (0.0,y/2.0,0.0);
                center: (0.0,0.0,0.0);
                bottom_hingepoint: (0.0,-y/2.0,0.0)
       end
  dynamics:;
  control:;
  interference:;
 end

composite linkage
 components
  h1: ball.and.socket;
  rod1: rod (1.0,4.0,1.0);
  base: rod (8.0,1.0,1.0)
 structure

  join rod1 to base with h1 matching
    (bottom_hingepoint  top_hingepoint);

 begin
  dynamics:;
  interference:;
 end;

configuration linkage_control
 components
  linkage: linkage
 control
   torque linkage.h1 linkage.torque
 state
   linkage.base: ((0.0, 0.0, 0.0) (1.0, 0.0, 0.0, 0.0) (0.0, 0.0, 0.0) (0.0, 0.0, 0.0));
   linkage.rod1: ((0.0, 2.5, 0.0) (1.0, 0.0, 0.0, 0.0) (0.0, 0.0, 0.0) (0.0, 0.0, 0.0))
```

**Fig. 1.** The Newton model definition for a pendulum.

symbolically define component parts and their interconnections. The simulation system will then create the necessary quantities and derive the constraint equations.

As a simple demonstration, we step through the derivation of the motion constraint equations for the pendulum example with the approach used in Newton. Newton supports incremental assembly of motion constraints. A skeletal representation is created for each primitive body. This representation includes values for object properties including density and, when necessary, material properties that enable computation of coefficients of restitution and friction. The representation contains a geometric model of the body that is used by Newton to compute properties such as object volume, mass, and inertia and for collision detection and contact classification. The representation also includes slots for kinematic and dynamic quantities including position, velocity, acceleration, force, and torque. Initially, each body inherits two simple equation schemata representing the Newton and Euler equations of motion. For body $i$ these equations have the form:

$$m_i \ddot{\mathbf{r}}_i = 0 \tag{1}$$

$$\mathbf{J}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \mathbf{J}_i \boldsymbol{\omega}_i = 0, \tag{2}$$

where

- $m_i$ is the mass of object $i$,
- $\ddot{\mathbf{r}}_i$ is the three dimensional acceleration vector for the center of mass of object $i$,
- $\mathbf{J}_i$ is the 3-by-3 inertia matrix for object $i$,
- $\boldsymbol{\omega}_i$ is the three dimensional angular velocity vector for object $i$, and
- $\dot{\boldsymbol{\omega}}_i$ is the three dimensional angular acceleration vector for object $i$.

Next, each hinge is considered and equations are derived to enforce the motion constraints imposed by the hinge. The spherical hinge linking the pendulum and base imposes a kinematic constraint that a distinguished point on one body, called the hinge point, be coincident with a hinge point on another body. If the hinge point for body $i$ is identified by a vector $\mathbf{c}_i$ from the center of mass $\mathbf{r}_i$ of body $i$ to the hinge, then a spherical hinge constraint requires that

$$\mathbf{r}_1 + \mathbf{c}_1 = \mathbf{r}_2 + \mathbf{c}_2, \tag{3}$$

where for compactness the pendulum and base are referenced by indices 1 and 2, respectively.

In Newton, position constraints are enforced indirectly by adding the second derivative (with respect to time) of the position constraint equation to the set of motion equations. For a spherical hinge, the second derivative equation ensures that the hinge points have the same acceleration. The motion constraint equation corresponding to the spherical hinge position constraint (3) is

$$\ddot{\mathbf{r}}_1 + \dot{\boldsymbol{\omega}}_1 \times \mathbf{c}_1 + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \mathbf{c}_1) = \ddot{\mathbf{r}}_2 + \dot{\boldsymbol{\omega}}_2 \times \mathbf{c}_2 + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \mathbf{c}_2). \tag{4}$$

The Newton and Euler equations for the connected bodies must also be modified to consistently reflect the constraint force $f_{hinge}$ required to satisfy the kinematic hinge constraint (4). A new force quantity is created to represent the constraint force and the Newton and Euler equations are augmented to give:

$$m_2 \ddot{\mathbf{r}}_2 = \mathbf{f}_{hinge}, \tag{5}$$

$$\mathbf{J}_2 \dot{\boldsymbol{\omega}}_2 + \boldsymbol{\omega}_2 \times \mathbf{J}_2 \boldsymbol{\omega}_2 = \mathbf{c}_2 \times \mathbf{f}_{hinge}, \tag{6}$$

$$m_1 \ddot{\mathbf{r}}_1 = -\mathbf{f}_{hinge}, \tag{7}$$

and

$$\mathbf{J}_1\dot{\boldsymbol{\omega}}_1 + \boldsymbol{\omega}_1 \times \mathbf{J}_1\boldsymbol{\omega}_1 = \mathbf{c}_1 \times -\mathbf{f}_{hinge.} \tag{8}$$

This general process can be used for a wide variety of kinematic constraints. Non-kinematic constraints, such as springs and dampers, can be handled in a similar fashion. If the action of gravity is to be modeled, a gravitational force term $m_i \mathbf{g}$ can be added to each object $i$'s Newton equation.

The motion constraint equations may be derived and represented in a variety of ways. Recently developed algorithms for evaluating sets of constraint equations can yield real-time simulation performance for moderately complex devices [9,14,30]. Many systems formulate the equations of motion once and compile them into a form suitable for quick evaluation. A disadvantage of this approach is that it is very difficult to modify the motion constraints during the course of a simulation. Newton represents the motion constraint equations symbolically. The pertinent equations are assembled and solved on each iteration. This facilitates inspection and incremental modification of motion constraints.

The method of analysis outlined above may introduce unacceptable errors into the simulation. Numerical error in the solution of the second order motion equations will lead to errors in positions and velocities. As this error accumulates the underlying position constraints may be significantly violated. Over the course of a simulation, the deterioration may seriously corrupt the integrity of the simulation. Many simulations system include a correction stage in the integration cycle to assure that positions and velocities do not drift too much.

The method of integration has an important influence on the accuracy and speed of a simulation. More economical methods use a state history to predict motions from previous positions and velocities. Tradeoffs between accuracy and speed can be made; in general, smaller time steps lead to more accurate results at the cost of greater computation and longer run times. Sophisticated schemes estimate the error in previous simulation steps and adaptively adjust the size of the time step to balance accuracy and speed. At the beginning of a simulation, there is no history on which to base predictions or error estimates and hence, simpler, more short-sighted methods must be used to get started. In Newton, a fourth-order Runge Kutta integration method that requires no history is used to initiate the simulation. After a sufficient history is built, Newton switches to the more economical fourth-order, adaptive-timestep Adams-Moulton integration method.

## 3. Events: Higher-Order Constraints

As physical objects move, conditions may occur that cause the relationships among bodies to change. For example, a contact hinge exists only while there is a compressive contact force between the connected objects. The support relation between a table and an object on its top surface behaves in this way. If an object rolls off the table or is lifted from the table, the support relationship no longer holds. The structural properties of joints may also limit the forces they can withstand. Excessive force on the joint may cause it to break.

The changing relations among physical objects can be modeled in simulation as state-dependent constraints. Contingent on the values of quantities, new constraints may be created and existing constraints may eliminated from the constraint set. A rule that constrains the composition of the constraint set is called a *higher-order constraint* in constraint programming [16]. These constraints on constraints are guarded by boolean expressions of the state variables. They may cause creation of new quantities, destruction of old quantities, or reformulation of the existing network, including the addition or deletion of higher-order

constraints.

For example, consider an object that comes to rest on a table. We model the new contact relation by establishing a temporary hinge. To simplify our example, we will assume the object touches the surface of the table at a single point. A new quantity is created to represent the constraint force at the hinge between the object and the table. A new equation is created to represent the constraint on the relative motions of the two bodies. A higher-order constraint is also created to monitor the conditions for continued existence of the hinge. When the conditions for existence no longer hold, the hinge constraint equation is removed from the constraint set, the constraint force quantity ceases to exist, and the hinge monitor is deleted.

The Newton simulator has a general mechanism for detecting and resolving exceptional events that require reformulation of the motion constraints. An event consists of a triggering condition, an isolation predicate, and a resolution method. At any point during the course of a simulation, every event is in either an active or inactive state. At the beginning of each integration step, the triggering conditions for every active event are tested. When the triggering condition is satisfied in the current state, an occurrence of the event has been detected. For some events, it is important to precisely determine the time of their occurrence. The validity of the resulting state may critically depend on isolating the instant at which the event happened. For example, when two rigid objects collide it is important to accurately determine the moment at which their surfaces first make contact to avoid interpenetration. The isolation predicate specifies the tolerance accepted in the timing of the event. The tolerance is specified as an acceptable error in some critical variable associated with the occurrence of the event. For many events, the error in the time of occurrence is used. However, other parameters may be appropriate for some events. For example, the time of collision can be isolated by specifying a tolerance on the separation or depth of penetration of the two objects.

After the time of occurrence of the event has been satisfactorily isolated the resolution method is executed. The actions taken during event resolution vary greatly with the type of event. Some kinds of events cause changes in the values of the state variables such as velocities. For example, collision events are resolved by formulating a set of impulse-momentum equations in a manner analogous to the formulation of the basic motion constraints. Solving this system yields instantaneous changes in velocities. Other types of events lead to reformulation of the set of motion constraints. Events corresponding to formation or breakage of temporary contacts are resolved by the addition or deletion of constraint equations.

Event resolution often leads to discontinuous changes in the values of state variables. The resolution method may be directly responsible for the discontinuity, as with collision resolution, or it may indirectly lead to a discontinuity by modifying the motion constraints. In either case, discontinuities can cause serious problems for some integration schemes. In general, techniques that rely on a state history are most sensitive to abrupt changes in state variables. To avoid the large errors that can result from violation of continuity assumptions, Newton alerts the integrator to the possible occurrence of discontinuities. The integrator responds by starting the integration process over again using the last consistent state as the initial condition. As with the first steps of the simulation, a fourth-order Runge Kutta method is used until a sufficient numbers of steps have been computed to switch to a fourth order Adams-Moulton method.

# 4. Control Programming

Goal-directed behavior is achieved by the controlled application of forces and torques. Virtual actuators must be introduced as the sources for these forces and torques. A virtual actuator applies a force or torque to a primitive object or about a hinge connecting a pair objects. We usually think of a virtual actuator as modeling a physical device such as an electric motor, a muscle, or a jet engine.

The simulator must provide a means to define control programs that determine the forces and torques to be applied by virtual actuators. As a simple example of a virtual actuator, we consider a single particle to which we attach a control force. A control program might specify a force function that will cause the particle to move along some desired trajectory. At each integration step, it is the control program's responsibility to assure that the actuator force is defined. The manner in which the control program determines actuator output has a critical influence on the veracity of the simulation and on the nature of the control functions that can be expressed. This section presents two alternatives for structuring the interface between control programs and other components of the simulator.

## 4.1. Loosely coupled control

One way to introduce control into a simulation is to require the programmer to define control processes that specify actuator outputs at each step of the integration cycle. The control processes must have access to state information and must assign values to actuators outputs contingent upon the current state. To prevent conflicting specification by two control processes and circular dependencies, the privilege of setting an actuator might be restricted to a single control process designated as owner of the actuator. Alternatively, multiple processes may be permitted to specify component forces that are summed to determine the total output of the actuator.

In this framework, the control component operates on the fringe of the integration cycle. Control processes are executed between integration steps. From the perspective of a control program, the dynamic model is part of a world that can be observed and influenced only by applying forces in reaction to changes in state variables. The only aspects of the model that can be directly specified by the control routines are actuator outputs.

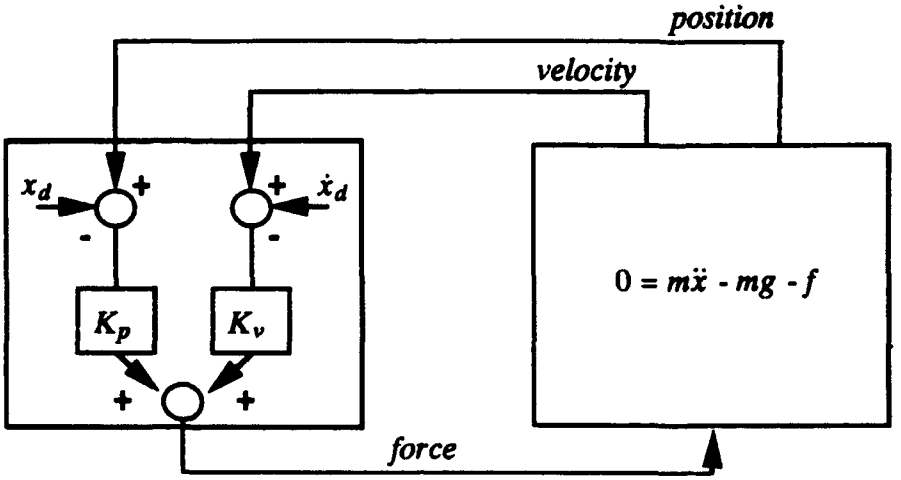
From the perspective of the integrator, control forces are treated as constants supplied by an external agent. In contrast, other second order quantities such as accelerations and hinge constraint forces are treated as unknowns to be determined by solving the motion equations.

The loosely coupled control regime emulates the relationship between control and mechanics in actual devices. To illustrate a loose coupling between the control and mechanical aspects of the simulation, we consider a one-dimensional particle actuated by a single control force. The components of the simulation are shown diagrammatically in Fig. 2.a. The position of the particle is given by $x$, the velocity by $\dot{x}$, and the acceleration by $\ddot{x}$. The motion of the particle is governed by a single motion equation:

$$f + mg = m\ddot{x} \tag{9}$$

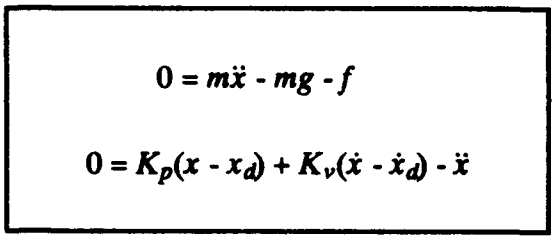
where,

- $f$ is the control force applied by the actuator,
- $m$ is the mass of the particle, and
- $g$ is the acceleration due to gravity.




**The Control Program** **The Simulated World**

(a) Loosely Coupled Control




**The Simulated World**

(b) Tightly Coupled Control

**Fig. 2.** The relationship between the control and analysis components of a simulator.

Given values for $m$ and $g$ and $f$, the integrator iteratively solves (9) to determine $\ddot{x}$. A control program supplies the value for $f$. To move the particle to a desired position $x_d$, our control program sets $f$ according to the a proportional, derivative control law:

$$f = K_p(x_d - x) + K_v(\dot{x}_d - \dot{x}) \tag{10}$$

There are several advantages to separating control processes from the process that enforces motion constraints due to physical properties. Transfer to applications is straightforward because the organization of the simulation emulates the physical control problem. The state information used in the simulated control program is also available in the physical control problem. The sensitivity to measurement errors can be tested in simulation by disturbing the values passed to the control component. Another advantage of loose coupling is that it is simple to enforce realistic limits on actuator output. The magnitude of the force determined by the control program can be tested to ensure that a plausible force is to be applied. If the force exceeds reasonable limits it may be clipped, the actuator may be disabled, or the event may be handled in some other predetermined fashion.

The disadvantage of loose coupling is that the controllable joint torques are usually related in only distant and complex ways to the behavior we want to achieve. Because of the physical coupling between objects in contact, a single force can influence the motion of many objects. It is very difficult to achieve even simple, controlled movements for articulated objects.

## 4.2. Tightly coupled control

In this approach control processes specify constraints on dynamic quantities. Control constraint equations are treated the same as constraint equations due to deterministic physical phenomena. As the simulation progresses, control processes add and delete constraint equations in response to changing circumstances.

The principal advantage of this approach is that desired quantities can be directly constrained. In the tightly coupled control regime, the one dimensional particle can be controlled by defining a new dynamic constraint equation that constrains the acceleration of the particle. As shown diagrammatically in Fig. 2.b, the control equation is combined with the dynamic equation derived from mechanical constraints. The system of equations is solved to determine the force required to satisfy the acceleration constraint.

Newton supports equation editing at the level of user-defined functions. The set of dynamic equations derived by the system can be augmented by linear, differential equations constraining second order quantities. There is no distinction between constraints arising from mechanical considerations and constraints due to control. The system of equations can include arbitrary linear constraints on accelerations, forces, and torques, and it may include references to arbitrary Lisp functions. Equations may be later retracted. A unique name is assigned to each equation when it is created to permit reference to it. Constraints may be expressed as vector or scalar equations.

Constraint equations can be defined to control composite quantities. For example, the acceleration of the center of mass of an articulated object could be constrained to influence the overall motion of an object. There must exist a set of actuators to account for the force and torque required to achieve the desired motion. If no actuator values can cause the acceleration, then the set of constraints is inconsistent and the equation solver will be unable to find a solution satisfying the constraints.

This points to one of the disadvantages of this less restrictive form of control programming. As is sometimes the case with constraint programming, it is easy to create inconsistent sets of constraints. Further, it is difficult to resolve the inconsistency to achieve the desired behavior. Inconsistencies of this sort were not possible in the loosely coupled approach. Because control routines could only set actuator forces, the control requirements were necessarily feasible, discounting possible limits on the magnitude of actuator output. Control routines could not cause the motion equations to be inconsistent. The more fundamental problem exists under both approaches, however. If the actuators are incapable of producing the desired motion, they will be incapable under any control regime. The limits of controllability are just manifested in different ways.

The freedom allowed to the programmer in the tightly coupled control regime can lead to other significant problems. In order to solve the constraint equations, the functions referenced in the control equations must be evaluated. If these functions are not restricted, then evaluation can disrupt the integrity of the constraint network. Functions may add or delete constraint equations as side effects during the solution process. The order in which functions are executed may have a radical influence on the results. The entire solution may be corrupted by modifying the underlying constraint network.

Side effects are not the only serious problem that can be caused by the introduction of user-defined functions in the constraint network. Functions that are discontinuous in the state variables can also lead to trouble. Many integration methods assume that quantities vary smoothly over time. Violations of this continuity assumption can lead to significant errors.

By placing restrictions on the functions referenced in constraint equations, we can preserve the integrity of the simulation. All functions must be continuous functions of state variables. Further, no side effects can occur within these functions.

The logic of the control program must be embedded in the event programming facility described above. At the beginning of the simulation, a user-defined initialization function adds an initial set of constraint equations to the constraint network and activates a collection of control events. Associated with each control event is a predicate that causes the event to be triggered. When, during the course of an integration step, an event is triggered, the event time is isolated and then the event is resolved.

As part of the event resolution, control constraint equations can be added to or deleted from the constraint network. To assure that the integrity of the physical model is not compromised, user-defined events should be allowed to add and delete only user-defined constraint equations.

The integrator must be notified when event resolution may lead to discontinuous in the values of state variables. The integrator can then restart the integration process using the values from the last state as initial values.

## 5. An Example: Hopping

We illustrate the constraint-based control method with a planar simulation of a hopping robot. The robot model, pictured in Fig. 3.a, consists of three links hinged with two revolute joints. The model emulates the torso, upper leg, and lower leg of an anthropoid. Actuators are located at joints labeled *knee* and *hip*. Repetitive hopping is controlled by a state dependent series of constraint equations. The event mechanism of Newton is used both to manage constraint equation editing and to model the mechanical interaction between the hopper and the ground.

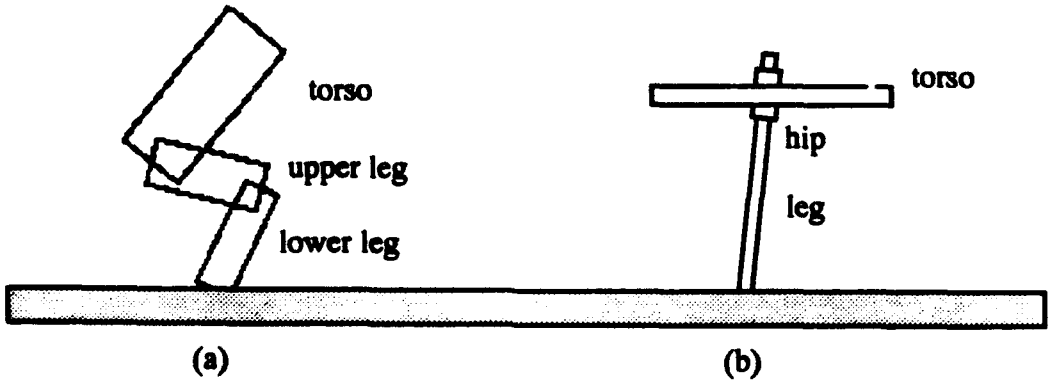



**Fig. 3.** Two hopper designs. The model on the left emulates an anthropoid. The model on the right is an idealization of the CMU one-legged hopper.

## 5.1 A Strategy for Hopping

Our control program for hopping is adapted from the three-part strategy used by Raibert to control the CMU hopper [10,11,22,23]. In an elegant series of experiments. Raibert and colleagues at Carnegie Mellon University (CMU) designed, built, and tested a variety of legged robots. The basic CMU hopper is illustrated in Fig. 3.b. It consisted of a large chassis connected to a telescoping leg with a revolute joint. An air cylinder in the leg gave it springiness along its longitudinal axis and provided a means of actuation. A second actuator could pivot the leg with respect to the trunk at the hip joint.

Raibert found that a wide variety of hoppers could be controlled with a simple, three-part control strategy. Control was decomposed into three subproblems:

(1) Posture Maintenance,
(2) Hop Height Adjustment, and
(3) Forward Speed Regulation.

It is immediately apparent that (for a planar hopper) the three kinematic parameters to be controlled outnumber the two controllable degrees of freedom. The problem was overcome by allocating actuators to control variables for a portion of time during each hop. The hopper cycled between *stance*, when the foot was in contact with the ground, and *flight*, when the hopper traveled ballistically through the air. A stable posture was maintained by adjusting the hip angle during stance to keep the trunk horizontal. Horizontal height was controlled by the leg actuator during stance. The leg acted as a spring, absorbing energy as it compressed during the first half of the stance interval and returning energy to the hopper as it expanding during the second half of the stance interval. The leg actuator injected sufficient energy during each stance period to compensate for energy lost during the collision with the ground.

Forward velocity was not explicitly controlled during stance. To achieve a constant forward velocity, the foot position was adjusted during the previous period of flight so that there would be no net acceleration over the stance period. Raibert observed that if the foot is placed such that the forward motion of the hopper will leave its center of mass directly over the foot when the leg spring is maximally compressed, then there will be an odd symmetry in the horizontal forces acting on the body over the stance interval. Backward acceleration in the first half of the stance period will be counterbalanced by forward acceleration during the second half period. The position of the foot, relative to the trunk at first contact, which led to this symmetric motion was called the neutral point. The hip actuator was devoted to control of leg orientation during the flight phase. Controlled acceleration and deceleration was achieved by displacing the foot position from the neutral point.

The CMU control strategy cannot be directly applied to the anthropomorphic hopper. Through a clever design, the CMU hopper decouples the posture and vertical velocity control variables during stance. Each control variable could be independently controlled by a separate actuator. The hip torque used to stabilize the orientation of the trunk did not influence forces along the leg and the force applied by the leg actuator to control vertical motion did not cause the trunk to rotate because its line of motion passed through the hip and center of mass of the trunk. No similar decoupling exists for the anthropomorphic hopper. The anthropomorphic robot must generate upward linear momentum in a coordinated push by the knee and hip actuators. The hopper's body must unfold from a crouched position during the second half of the stance period.

The structure of the anthropomorphic hopper forces us to redefine the control variables. Since trunk rotation is required to hop, stable posture cannot be maintained by keeping the trunk horizontal. Instead of trunk orientation, we focus on the angular momentum of

the hopper as a criterion for stability. Angular momentum is preserved during the period of flight following liftoff. If, during flight, the body rotates too far in either the forward or backward direction, limitations on the range of joint angles can prevent placement of the foot underneath the body and lead to an uncomfortable collision between the torso and ground. In animal hopping, the knee and hip joints dissipate and absorb kinetic energy when the falling body lands. The three-part control strategy for hopping uses the hopper's leg in a similar way. To accomplish this, it is critical to keep the leg underneath the torso. Our goal is to leave the ground with a desired velocity, and with no angular momentum.

## 5.2 Control of One Hop

Our approach concentrates on the interaction between the object and its surroundings at points of contact. It is only through reactions at contact points that a hopper can alter its momenta. The nature of contact relations places strict constraints on the dynamics of control problems. Our control program is expressed as dynamic constraints on the external forces and torques applied to the hopper by pushing against surfaces in contact.

The free-body diagram for the robot is shown in Fig. 4.

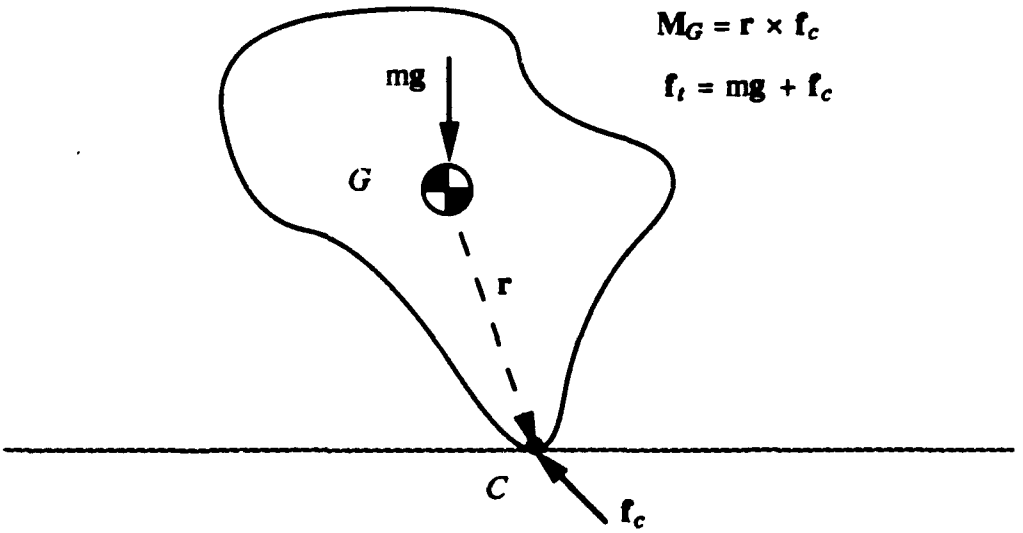



**Fig. 4.** The hopper free-body diagram.

Two external forces act on the composite robot body. The weight, $m\mathbf{g}$, acts through the center of mass and is directed downward along the $-\hat{Y}$ axis of the inertial coordinate system. The other force, $\mathbf{f}_c$, represents the reaction force of the ground and acts at the point of contact.

The overall instantaneous motion of the composite body is determined by the sum of external forces and moments acting on the body about $G$. The resultant force on the body, $\mathbf{f}_t$ is the sum the force caused by gravity and the reaction force:

$$m\mathbf{a}_G = \mathbf{f}_t = m\mathbf{g} + \mathbf{f}_c \tag{11}$$

where $\mathbf{a}_G$ is the inertial acceleration of the center of mass. Since the force of gravity passes through the center of mass, the only moment on the body about the mass center is that caused by the reaction force at the point of contact:

$$\dot{\mathbf{H}}_G = \mathbf{M}_G = \mathbf{r} \times \mathbf{f}_c, \tag{12}$$

where,

$\mathbf{M}_G$ is the resultant external moment about the center of mass of the composite body and

$\dot{\mathbf{H}}_G$ is the time-derivative of the angular momentum of the body about the center of mass, $\mathbf{H}_G$.

The overall instantaneous motion of the composite body during stance is completely determined by (11) and (12).

We will call the vector $\mathbf{r}$ the virtual leg [22,24]. The virtual leg determines the relationship between the contact force and moment on the body about the mass center. Note that this relationship is independent of the internal structure of the composite body. The virtual leg allows us to treat an arbitrary body as a lumped mass connected to the ground by a massless leg. By expressing the equations of motion in terms of the contact force, we can ignore the actuator forces and torques that contribute to the reaction force. This permits us to focus on the interaction between the body and its environment and will allow us to express control processes independent of device actuation or body structure.

The contact force is decomposed into components parallel and orthogonal to the virtual leg. We introduce constraint equations to control the two components of the contact force. The component of the contact force parallel to $\mathbf{r}$ is constrained to behave as a linear spring along the leg axis. Hence, the spring force may be expressed as

$$\mathbf{f}_c \cdot \frac{\mathbf{r}}{|\mathbf{r}|} = k_l(|\mathbf{r}| - r_s) \tag{13}$$

where $|\cdot|$ is the vector norm operator, $k_l$ denotes the spring stiffness, and the scalar $r_s$ is the resting length of the spring.

The component of the contact force perpendicular to the virtual leg is used to control the resultant moment about $G$ acting on the body. The value of this moment is constrained to reduce the difference between the angular momentum of the body about its mass center, $\mathbf{H}_G$, and the desired angular momentum of the body about its mass center, $\tilde{\mathbf{H}}_G$:

$$\mathbf{M}_G = \mathbf{r} \times \mathbf{f}_c = k_h(\tilde{\mathbf{H}}_G - \mathbf{H}_G). \tag{14}$$

The constant of proportionality, $k_h$, determines the rate at which corrections are made. The angular momentum about the mass center of an articulated body consisting of $n$ rigid links is defined by

$$\mathbf{H}_o = \sum_{i=1}^{n} \left[ \mathbf{I}_i \boldsymbol{\omega}_i + (\mathbf{c}_i \times m_i \mathbf{v}_i) \right] \tag{15}$$

where

$\mathbf{I}_i$ is the inertia tensor of link $i$ about its center of mass,
$\boldsymbol{\omega}_i$ is the angular velocity of link $i$,
$m_i$ is the mass of link $i$,
$\mathbf{v}_i$ is the velocity of the center of mass of link $i$, and
$\mathbf{c}_i$ is the vector from the center of mass of the composite body to the center of mass of link $i$.

The two constraint equations (13) and (14) have a simple geometric interpretation. The component of force along the virtual leg acts as a spring pushing the body away from the ground. The component orthogonal to the virtual leg moderates the direction of the push and applies a torque to the body about the mass center. If the pushing is directed to the left of the center of mass in Fig. 4, a clockwise torque is applied to the body.

## 5.3 Control of a Hopping Sequence

A series of hops begins with the hopper positioned above the ground surface with all body segments vertical. As the hopper falls, the foot is positioned for the first hop. The desired foot position was determined using Raibert's approximation of the neutral point. The virtual leg is also shortened as it falls to avoid landing in a singular configuration. An event is activated to detect contact between the foot and ground. The event is triggered when the foot penetrates the surface of the ground. The isolation predicate for the contact event specifies the acceptable error in the time of first contact. When the time of contact has been satisfactorily isolated the event is resolved by:

(1) Solving impact equations for an inelastic collision between the foot and ground and adjusting velocities of the hopper's limbs accordingly,[1]

(2) Deleting the control constraint equations used to position the foot and adding constraint equations for stance,

(3) Creating a revolute hinge between the lower leg and the ground,

(4) Deactivating the contact event and activating an event to detect liftoff, and

(5) Alerting the simulator to the possible discontinuities in dynamic quantities resulting from the collision and changes in control equations.

During the stance period that follows, constraint equations (13) and (14) control the force along the virtual leg and the moment on the hopper, respectively. Energy is injected by gradually increasing the resting length of the virtual leg to compensate for energy lost at impact [25].

The simulation continues uninterrupted until the constraint force at the temporary hinge with the ground becomes tensile. At this point the liftoff event is triggered. The isolation predicate accepts the time of first detection as the time of liftoff. The liftoff event is resolved by:

(1) Breaking the hinge between the lower leg and the ground,

(2) Deleting the control constraint equations for stance and adding the foot positioning constraint equations,

(3) Deactivating the liftoff event and activating the contact event to detect the next touchdown,

(4) Alerting the simulator to the possible discontinuities in dynamic quantities resulting from the collision and changes in control equations.

This begins the hopping cycle over again.

The process outlined above was slightly modified to avoid premature liftoff and touchdown. Brief transition periods were inserted to allow loading of the leg at touchdown and to gain height after liftoff. When the leg made first contact with the ground the control

1 The impulse-momentum equations are automatically derived from the geometry of the foot-ground collision and the dynamics of the hopper.

equations for stance were instantiated. However, the event to detect liftoff was activated only after a short waiting period. This allowed the contact force to become sufficiently large that small numeric errors would not cause the liftoff event to trigger prematurely. The delay at liftoff was needed to so that the foot would not touch the ground as it was swung forward for the next touchdown. The knee and hip joints were held in place for a short period as the hopper gained height.

One hop from a simulation of the anthropomorphic hopper is shown in Fig. 5.a. An important advantage of our formulation is that the constraints on the contact force during stance apply to a wide variety of different hoppers. To demonstrate this device independence, we simulated hopping with a model similar to the CMU hopper. A hop from a simulation of the CMU hopper is shown in Fig. 5.b. The control programs for the two hoppers differed only in the adjustment of leg lengths during flight.

## 6. Beyond Basic Hopping

Constraint-based control programming permits a style of programming comparable to *object-level* programs for robot manipulation. An object-level language specifies robot operations by defining the desired state of the object to be manipulated [1,15,17,20,21,27,32]. The robot actions required to bring about the necessary changes to the object are determined by lower levels of the robot system. In a similar way, the contact constraints used to control the hopper during stance treat the hopper as an object to be pushed or spun. The actuator values required to achieve a desired external force or torque are determined by lower level processes.

The usefulness of contact constraint programming goes beyond basic hopping. The one-legged robot considered above has contact with the ground during stance at a single point. The point contact permits only a single constraint force to be applied to the hopper. For this reason, the external force and moment on the hopper are intrinsically coupled. More complex interactions afford a rich set of alternatives for control required to achieve many behaviors. For example, a hopper with a sizable foot touching the ground shares a plane of contact with the support surface. With planar contact, the hopper can independently control the external force and moment during stance. We've demonstrated the increased controllability by simulating a flip with no change in horizontal velocity. Because the external force and moment are decoupled, the hopper can jump vertically with sufficient momentum to rotate through a complete circle. Frames from an animation of a flip are shown in Fig. 6. We believe that the contact constraint analysis will prove useful in understanding a wide variety of locomotion tasks.

The difficulty in controlling the motion of complex, articulated figures has impeded progress in physically-based animation. We believe the constraint-based control paradigm offers a rich, expressive tool for directing purposeful actions of mechanical models.

## Acknowledgement

We would like to thank James G. Andrews for his assistance with the dynamic analysis and many valuable suggestions. This work was supported by National Science Foundation Grants IRI-8808896 and IRI-9006137, Office of Naval Research Grants ONR 00014-88K-0632 and N00014-89-J-1946, and by the Advanced Research Projects Agency of the Department of Defense under ONR Contract N00014-88-K-0591.

**References**


1. Ambler, A., Cameron, S. A., and Corner, D. F., "Augmenting the RAPT robot language," in *Languages for Sensor-based Control in Robotics*, ed. Hormann, pp. 305-316, Springer-Verlag, New York, NY, 1987.
2. Armstrong, W. W. and Green, M. W., "The dynamics of articulated rigid bodies for purposes of animation," *The Visual Computer*, vol. 1, pp. 231-240, 1985.
3. Armstrong, W. W., Green, M. W., and Lake, R., "Near-real-time control of human figure models," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 52-61, June 1987.
4. Badler, N. I., Manoochehri, K. H., and Walters, G., "Articulated figure positioning by multiple constraints," *IEEE Computer Graphics and Applications*, pp. 28-38, June 1987.
5. Badler, N. I., Barsky, A., and Zeltzer, D., *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.
6. Calvert, T. W., "The challenge of human figure animation," *Graphics Interface '88*, pp. 203-210, 1988.
7. Chace, M., "Methods and experience in computer aided design of large displacement mechanical systems," in *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, ed. E. J. Haug, vol. 9, NATO ASI Series F: Computer and System Sciences, Springer-Verlag, 1984.
8. Cremer, J., "An Architecture for General Purpose Physical Simulation -- Integrating Geometry, Dynamics, and Control," Ph.D. Thesis, TR 89-987, Cornell University, April, 1989.
9. Haug, E. J., *Computer Aided Kinematics and Dynamics of Mechanical Systems*, 1, Allyn and Bacon, Boston, MA, 1989.
10. Hodgins, J. K. and Raibert, M. H., "Biped Gymnastics," *The International Journal of Robotics Research*, vol. 9, no. 2, pp. 115-132, April 1990.
11. Hodgins, J. K. and Raibert, M. H., "Adjusting step length for rough terrain locomotion," *IEEE Transactions of Robotics and Automation*, vol. 7, no. 3, pp. 289-298, June 1991.
12. Hoffmann, C. M. and Hopcroft, J. E., "Simulation of physical systems from geometric models," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 194-206, June 1987.
13. Hollars, M. G., Rosenthal, D. E., and Sherman, M., in *SD/FAST User's Manual*, Symbolic Dynamics, Inc., Mountain View, CA, 1990.
14. Kane, T. R. and Levinson, D. A., *Dynamics: Theory and Applications*, McGraw-Hill Book Company, New York, 1985.
15. Latombe, J. C., Laugier, C., Lefebvre, J. M., Mazer, E., and Miribel, J. F., "The LM robot programming system," in *Robotics Research - The Second International Symposium*, ed. H. Inoue, pp. 377-391, MIT Press, Cambridge, MA, 1985.
16. Leler, W., *Constraint Programming Languages*, Addison-Wesley Publishing Company, Reading, MA, 1988.
17. Lozano-Perez, T., "Robot Programming," *Proceedings of the IEEE*, vol. 71, no. 7, July 1983.

18. Magnenat-Thalmann, N. and Thalmann, D., in *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo, Japan, 1985.

19. Magnenat-Thalmann, N. and Thalmann, D., in *State-of-the-art in Computer Animation: Proceedings of Computer Animation '89*, Springer-Verlag, Tokyo, Japan, 1989.

20. Popplestone, R., Ambler, A., and Bellos, I., "An interpreter for a language for describing assemblies," *Artificial Intelligence*, vol. 14, pp. 79-107, 1980.

21. Popplestone, R. and Ambler, A., "A language for specifying robot manipulations," in *Robotic Technology*, ed. A. Pugh, pp. 125-141, Peter Peregrinus, London, 1983.

22. Raibert, M. H., *Legged Robots That Balance*, The MIT Press, Cambridge, MA, 1986.

23. Raibert, M. H., Chepponis, M., and Brown, H. B. Jr., "Running on four legs as though they were one," *IEEE Journal of Robotcs Research*, vol. 2, 1986.

24. Sutherland, I. E. and Ullner, M. K., "Footprints in the asphalt," *International Journal of Robotics Research*, vol. 3, pp. 29-36, 1984.

25. Sznaier, M. and Damborg, M. J., "An Adaptive Conroller for a One-Legged Mobile Robot," *IEEE transactions on Robotics and Automation*, vol. 5, no. 2, pp. 253-259, April 1989.

26. Upson, C., Barr, A., Reeves, B., Wolff, R., and Wolfram, S., "The physical simulation and visual representation of natural phenomena," *SIGGRAPH '87 Conference Proceedings*, pp. 335-336, 1987.

27. Volz, R., "Report of the robot programming language working group: NATO workshop on robot programming languages," *IEEE Journal of Robotics and Automation*, vol. 4, no. 1, pp. 86-90, Feb 1988.

28. Wilhelms, J., "Using dynamic analysis for realistic animation of articulated figures," *IEEE Computer Graphics and Applications*, vol. 7, no. 6, pp. 12-27, June 1987.

29. Witkin, A., Gleicher, M., and Welch, W., "Interactive dynamics," *Computer Graphics*, vol. 24, no. 2, pp. 11-21, March 1990.

30. Wittenburg, J., *Dynamics of Systems of Rigid Bodies*, B. G. Teubner Stuttgart, 1977.

31. Zeltzer, D., "Motion control techniques for figure animation," *IEEE Computer Graphics and Applications*, pp. 53-59, Nov 1982.

32. Zeltzer, D., "Task-level graphical simulation: Abstraction, representation, and control," in *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, ed. D. Zeltzer, pp. 3-33, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

*NOTE to Publisher: The captions for the color photographs are included below. The pictures are identified as Fig. 5.a, Fig. 5.b, and Fig. 6 in the text and in the captions. The photographs are labeled on back with the correct Fig. number.*

**Fig. 5.** Animation sequences for two hoppers. In both figures, a series of frames from a single hop is shown. The hopper in the 5.a is articulated like a human with links similar to the torso, upper leg, and lower leg. The hopper in the 5.b modeled after the one-legged CMU hopper.

**Fig. 6.** Simulation results for the human-like hopper in surface contact with the ground during stance. The surface contact enables foward velocity and angular momentum to be decoupled. This allows the hopper to jump straight upward with sufficient angular momentum to flip.

FIG. 5.a

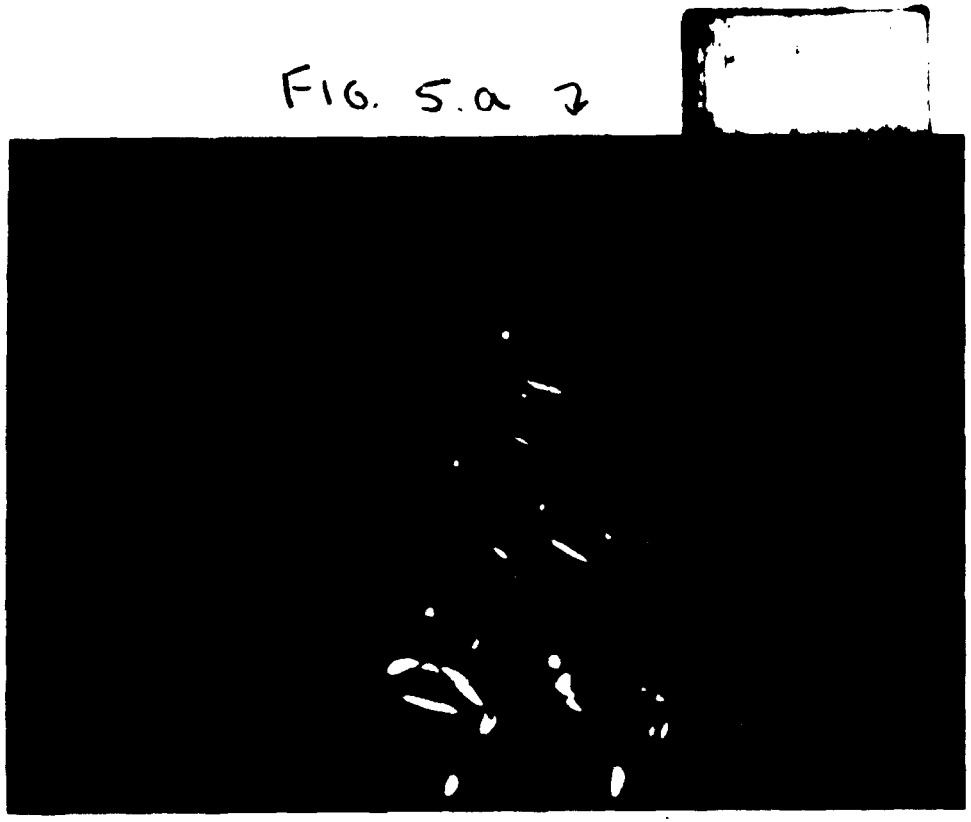
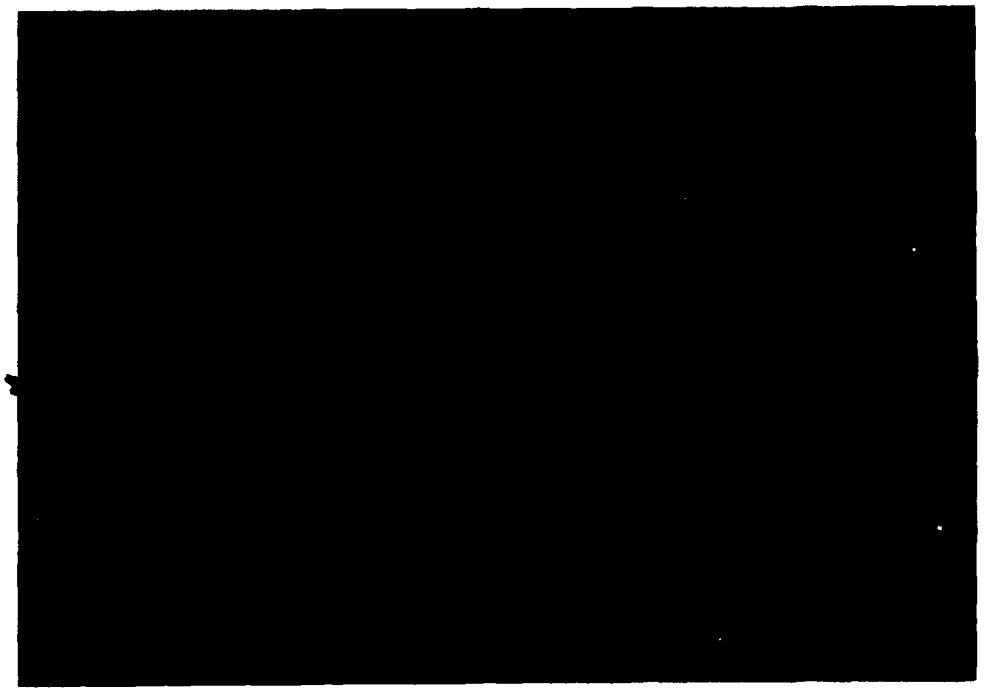


FIG 5.b



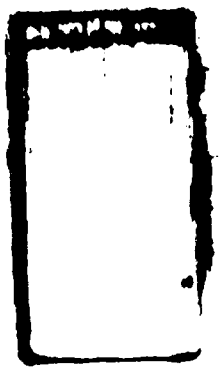


FIG 6